AMATH 482 Homework 3

Wenxuan Liu

February 24, 2021

Abstract

We have four sets of three videos filming the same can doing simple harmonic motion in vertical direction. The three cameras are in three angles, so they give different versions of same information. Our four tasks are analyzing the can's motion using PCA for the four sets of videos. For the first task, there is only vertical movement; for the second task, camera is shaking and can is moving vertically; for the third task, the can is moving in both horizontal and vertical directions; for the fourth task, the can moves in vertical direction while spinning. For each task, we extract the x-y position of the can and form a six-dimensional matrix (3 videos * 2 dimensions each) that records the can position. We do PCA on it and the singular values can tell us the principle component of the movement.

1 Introduction and Overview

We have four tasks of analyzing can movement using three videos in each of the following cases:

1. Ideal Case & Noisy Case: the can moves vertically, so we want to extract the x-y position of the can in each frame. We do this by looking for the flash light placed above the can. The brightest point of the image is the position we want. We also want to mask areas of the frames that we don't need in order to get cleaner position data. Then we want to do project the data on the principal components to see what is captured in each direction.
2. Horizontal Displacement & Horizontal Displacement and Rotation: the can not only moves up and down, but also moves horizontally/rotates. Thus, after doing PCA, we want different components capture different movements of the can. In the last case, it is harder to extract the position of the can because the flashlight often disappears; thus, we track the red color on the can instead.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

As we learned from our text book (lecture 11) [1], The singular value decomposition of the matrix A is:

$$A = U\Sigma V^* \tag{1}$$

Where where $U \in \mathbf{R}^{m\times m}$ and $\Sigma \in \mathbf{R}^{n\times n}$ are unitary matrices, and $V \in \mathbf{R}^{m\times n}$ is diagonal. Based on this idea, multiply by A on the left can be decomposed to: rotate by V^* to align V_n with the axes, stretch by Σ in each direction, and rotate back by U.
Unlike Eigenvalue Decomposition, any matrix has SVD. The singular values are uniquely determined and are always non-negative real numbers.The singular values are ordered from largest to smallest along the diagonal of Σ.
The singular values tell us what is the most important component of the matrix.

To calculate SVD, we use the following equation:

$$\begin{aligned} A^*A &= (U\Sigma V^*)^*(U\Sigma V^*) \\ &= V\Sigma U^* U\Sigma V^* \\ &= V\Sigma^2 V^* \\ A^*AV &= V\Sigma^2 \end{aligned}$$

So that the columns of V are eigenvectors of A^*A with eigenvalues given by the square of the singular values.

$$\begin{aligned} AA^* &= (U\Sigma V^*)(U\Sigma V^*)^* \\ &= U\Sigma V^* V\Sigma U^* \\ &= U\Sigma^2 U^* \\ A^*AU &= U\Sigma^2 \end{aligned}$$

So that the columns of U are eigenvectors of AA^*

2.2 Low-Dimensional Approximations

From text book (lecture 11) [1], we learned Low-Dimensional Approximations of matrix, which is an application of SVD. Given a matrix A with rank r, the best rank N approximation of A is given by:

$$A_N = \sum_{j-1}^{N} \sigma_j u_j v_j^* \tag{2}$$

3 Algorithm Implementation and Development

3.1 Ideal Case & Noisy Case

1. First we load the videos into MatLab. Then, we resize the video by cutting off from beginning or end to make sure the three videos align in time.

2. Loop through each frame of the three videos and extract position of the can. We first turn the video into gray and then mask parts we don't want. The position is given by the brightest points on the frame, and we average the indices of maximum values to get the position of the can. Sudo code:

Algorithm 1: Extract Can Position Algorithm

```
Define empty position array
for j = 1 : num_frame do
    Take the i^th frame out and turn it to gray
    Mask the parts we don't need
    Find the maximum indices and average over them
    Store the x,y coordinates
end for
```

3. Repeat 1, 2 for the rest videos and form a six dimensional matrix of the form:

$$[cam\_1\_X; cam\_1\_Y; cam\_2\_X; cam\_2\_Y; cam\_3\_X; cam\_3\_Y]$$

We subtract the mean from each row of displacement matrix. Then we perform SVD on the above matrix and make projections with SVD to see how the principal components of original data are captured.

3.2 Horizontal Displacement

1. First we load the videos into MatLab. Then, we resize the video by cutting off from beginning or end to make sure the three videos align in time.

2. By observing, we notice that the first video has two problems: flash light disappears for a few seconds, and the horizontal movement is opposite to the other two frames. To solve the first problem, we keep only the red component of the video and look for the reddest part; we do it because there is red object on top of the can and not elsewhere. We solve the second problem by mirror the X coordinate, so the X-direction movement align with the other two videos. Sudo code:

Algorithm 2: Extract Can Position Algorithm

```
Define empty position array
for j = 1 : num_frame do
    Take the i^th frame out and turn it to gray
    Mask the parts we don't need
    Mask the green, blue component of the frame
    Find the maximum indices and average over them
    Store the (frame_length - x, y) coordinates
end for
```

3. Repeat 1, 2 for the rest videos and form a six dimensional matrix of the form:

$$[cam\_1\_X; cam\_1\_Y; cam\_2\_X; cam\_2\_Y; cam\_3\_X; cam\_3\_Y]$$

We subtract the mean from each row of displacement matrix. Then we perform SVD on the above matrix and make projections with SVD to see how the principal components of original data are captured.

3.3 Horizontal Displacement and Rotation

We repeat the process in 3.2 Horizontal Displacement, however, we add one more mask to the frame. In this case, we don't have the flash light, and the back ground is too bright to track the reddest pixels in the old way. To solve the problem, we use the Color Thresholder app provided by MatLab. The generated mask function is in Appendix B. This function takes a video frame and mask everything that is not red enough, then return the frame.

4 Computational Results

4.1 Ideal Case

The first picture shows the position of the can over time (See Figure 1). The second picture tells us the first component is dominant feature of the movement, and third picture is the projection of the original data on the principle components, see Figure 2. Through the first picture, we can see that all three cameras captures the vertical simple harmonic motion, and the movement in other directions is just noise. The largest energy of diagonal variance indicates that there is a dominant component in the displacement. Figure three captures the up and down movement of the can, which is exactly what we want; however, due to the nonalignment of cameras, there is a second largest component capturing same movement but shifted. The third principal component is just capturing some noise.

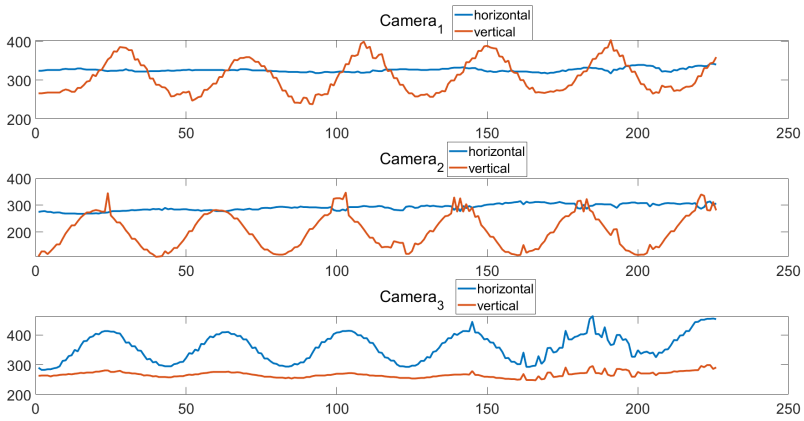
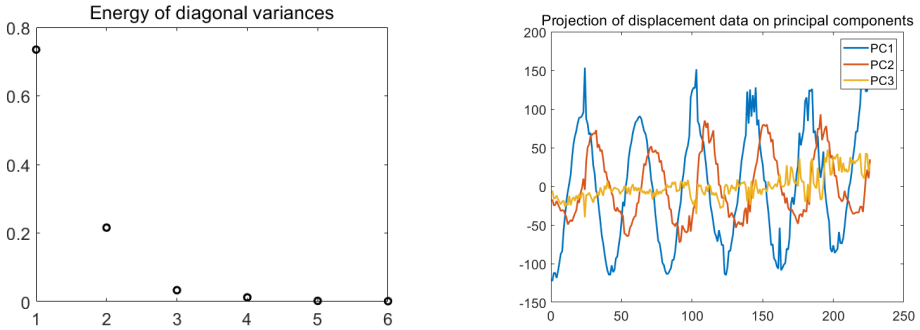


Figure 1: Can Movement From 3 Cams



(a) Energies

(b) Projection on CPs

Figure 2: PCA Analysis

4.2 Noisy Case

The result is as same as 4.1 Ideal Case, but more noise is introduced, so the principal components 3, 4 are larger than the first case.

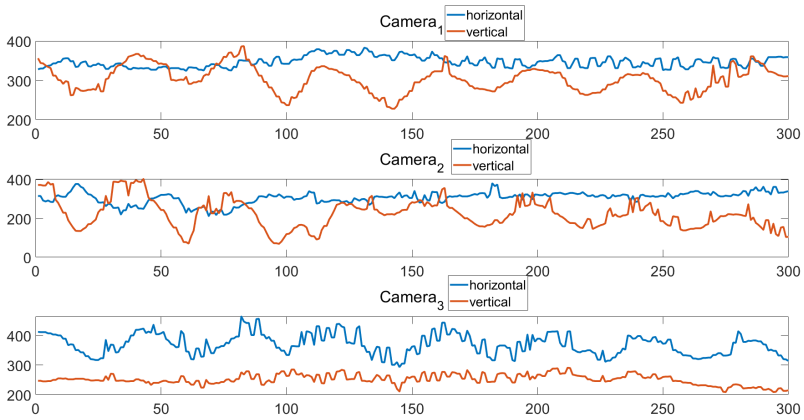
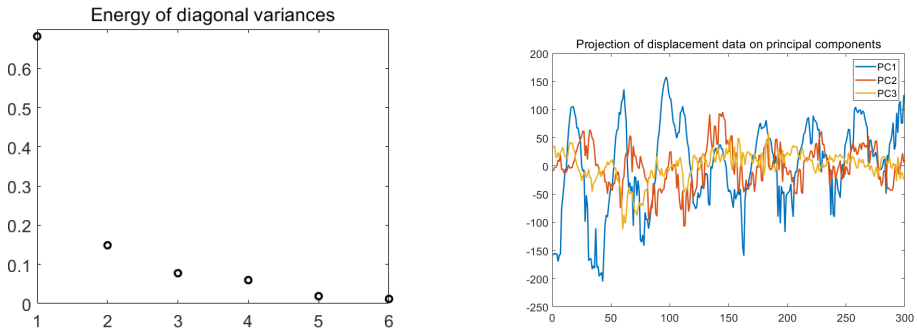


Figure 3: Can Movement From 3 Cams



(a) Energies (b) Projection on CPs

Figure 4: PCA Analysis

4.3 Horizontal Displacement

There is simple harmonic motion in both vertical and horizontal directions, see Figure 5 for the can position. By projecting on first principal component, we can capture the vertical movement, and second principal component projection is able to restore the horizontal movement. The projections clearly show the movement in two directions.

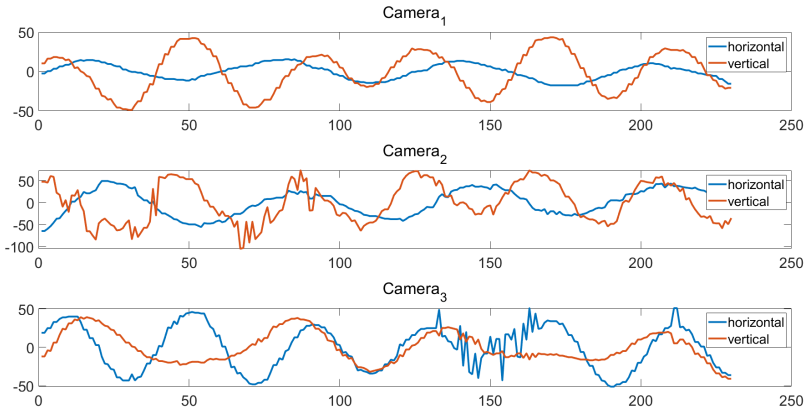
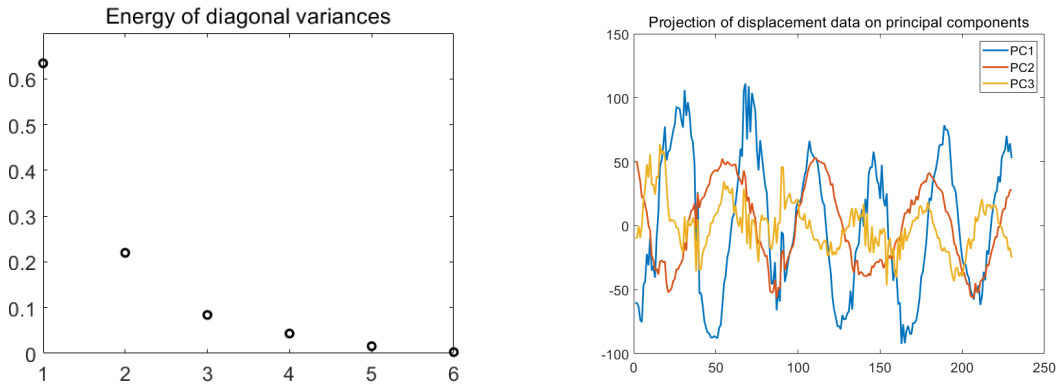


Figure 5: Can Movement From 3 Cams



(a) Energies (b) Projection on CPs

Figure 6: PCA Analysis

4.4 Horizontal Displacement and Rotation

The vertical movement is not perfectly captured from cam_2, and the displacement caused by the rotation is not clearly visible. The result of diagonal variances and projection are not showing us the second principle component that we want to see.

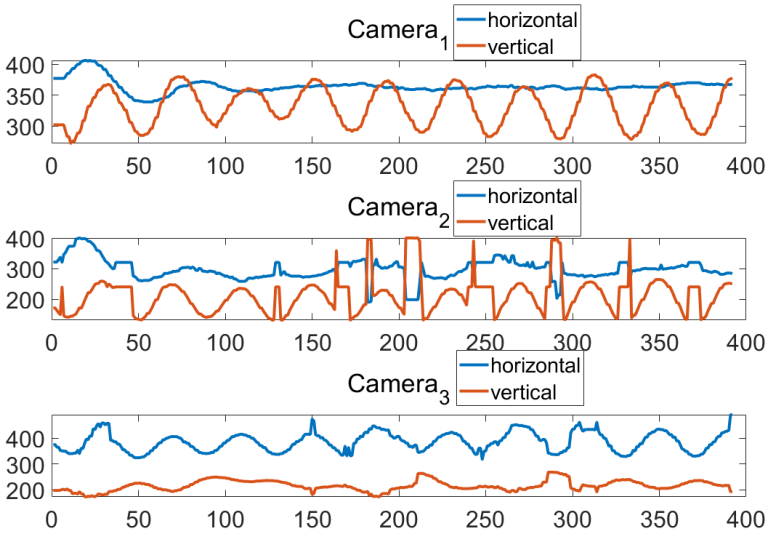
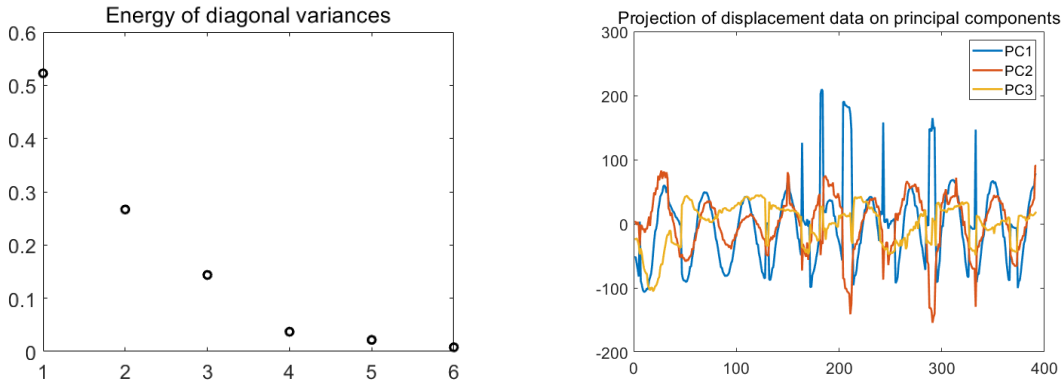


Figure 7: Can Movement From 3 Cams



(a) Energies

(b) Projection on CPs

Figure 8: PCA Analysis

5 Summary and Conclusions

After a series of principle component analysis on the four cases, we can get the principle components of each type of motion. With the CPs, we can identify the movements in horizontal or vertical directions. The result of the rotation case is not satisfying mainly because of difficulty of position extraction and excessive noise.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `I = rgb2gray(RGB)` converts the truecolor image RGB to the grayscale image I. The rgb2gray function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance. If you have Parallel Computing Toolbox™ installed, rgb2gray can perform this conversion on a GPU.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `D = diag(v)` returns a square diagonal matrix with the elements of vector v on the main diagonal.
- `[row,col] = ind2sub(sz,ind)` returns the arrays row and col containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size sz. Here sz is a vector with two elements, where sz(1) specifies the number of rows and sz(2) specifies the number of columns.
- `HSV = rgb2hsv(RGB)` converts the red, green, and blue values of an RGB image to hue, saturation, and value (HSV) values of an HSV image.
- `B = repmat(A,n)` returns an array containing n copies of A in the row and column dimensions. The size of B is size(A)*n when A is a matrix.
- `imshow(I)` displays the grayscale image I in a figure. imshow uses the default display range for the image data type and optimizes figure, axes, and image object properties for image display.
- `C = bsxfun(fun,A,B)` applies the element-wise binary operation specified by the function handle fun to arrays A and B.

Appendix B MATLAB Code

MATLAB code that produce the outcomes discussed in this paper.

```
%% task1
% Clean workspace
clear all; close all; clc

load('cam1_1.mat')
numFrames1_1 = size(vidFrames1_1,4);

positions1_1 = zeros(2, numFrames1_1);
for j = 1:numFrames1_1
    X = rgb2gray(vidFrames1_1(:,:,:,j)); %y,x
    X((1:480),(1:250)) = zeros(480,250);
    X((1:480),(451:640)) = zeros(480,190);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions1_1(:, j) = [bigx; bigy];
end

load('cam2_1.mat')
numFrames2_1 = size(vidFrames2_1,4);

positions2_1 = zeros(2, numFrames2_1-58);
for j = 20:245
    X = rgb2gray(vidFrames2_1(:,:,:,j)); %y,x
    X((1:480),(1:220)) = zeros(480,220);
    X((1:480),(421:640)) = zeros(480,220);
```

```
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions2_1(:, j-19) = [bigx; bigy];
end

load('cam3_1.mat')
numFrames3_1 = size(vidFrames3_1,4);

positions3_1 = zeros(2, numFrames3_1-7);
for j = 7:numFrames3_1
    X = rgb2gray(vidFrames3_1(:,:,:,j)); %y,x
    X((1:230),(1:640)) = zeros(230,640);
    X((351:480),(1:640)) = zeros(130,640);
    X((1:480),(1:170)) = zeros(480,170);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions3_1(:, j-6) = [bigx; bigy];
end
% first plot three cameras' x and y motion

position_1 = [positions1_1; positions2_1; positions3_1];
plotEverything(position_1, 226)
%% task2
% Clean workspace
clear all; close all; clc

load('cam1_2.mat')
numFrames1_2 = size(vidFrames1_2,4);

positions1_2 = zeros(2, numFrames1_2-14);
for j = 15:numFrames1_2
    X = rgb2gray(vidFrames1_2(:,:,:,j)); %y,x
    X((1:480),(1:250)) = zeros(480,250);
    X((1:480),(451:640)) = zeros(480,190);
    X((1:200),(1:640)) = zeros(200,640);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions1_2(:, j-14) = [bigx; bigy];
end

load('cam2_2.mat')
numFrames2_2 = size(vidFrames2_2,4);

positions2_2 = zeros(2, numFrames2_2-56);
for j = 1:numFrames2_2-56
    X = rgb2gray(vidFrames2_2(:,:,:,j)); %y,x
    X((1:480),(1:180)) = zeros(480,180);
    X((1:480),(421:640)) = zeros(480,220);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
```

```
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions2_2(:, j) = [bigx; bigy];
end


load('cam3_2.mat')
numFrames3_2 = size(vidFrames3_2,4);

positions3_2 = zeros(2, numFrames3_2-27);
for j = 15:numFrames3_2-13
    X = rgb2gray(vidFrames3_2(:,:,:,j)); %y,x
    X((1:200),(1:640)) = zeros(200,640);
    X((331:480),(1:640)) = zeros(150,640);
    X((1:480),(1:220)) = zeros(480,220);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions3_2(:, j-14) = [bigx; bigy];
end
position_2 = [positions1_2; positions2_2; positions3_2];
plotEverything(position_2, 300)

%% task3
% Clean workspace
clear all; close all; clc

load('cam1_3.mat')
numFrames1_3 = size(vidFrames1_3,4);

positions1_3 = zeros(2, numFrames1_3-9);
for j = 10:numFrames1_3
    X = vidFrames1_3(:,:,:,j); %y,x
    X((1:480),(1:250)) = zeros(480,250);
    X((1:480),(451:640)) = zeros(480,190);
    X((1:200),(1:640)) = zeros(200,640);
    %keep red component
    X(:,:,(2:3)) = zeros(480,640, 2);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions1_3(:, j-9) = [480-bigx; bigy];
end

load('cam2_3.mat')
numFrames2_3 = size(vidFrames2_3,4);

positions2_3 = zeros(2, numFrames2_3-51);
for j = 1:numFrames2_3-51
    X = rgb2gray(vidFrames2_3(:,:,:,j)); %y,x
    X((1:480),(1:180)) = zeros(480,180);
    X((1:480),(421:640)) = zeros(480,220);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
```

```
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions2_3(:, j) = [bigx; bigy];
end

load('cam3_3.mat')
numFrames3_3 = size(vidFrames3_3,4);

positions3_3 = zeros(2, numFrames3_3-7);
for j = 1:numFrames3_3-7
    X = rgb2gray(vidFrames3_3(:,:,:,j)); %y,x
    X((1:200),(1:640)) = zeros(200,640);
    X((331:480),(1:640)) = zeros(150,640);
    X((1:480),(1:220)) = zeros(480,220);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions3_3(:, j) = [bigx; bigy];
end

position_3 = [positions1_3; positions2_3; positions3_3];
plotEverything(position_3, 230)
%% task4
% Clean workspace
clear all; close all; clc

load('cam1_4.mat')
numFrames1_4 = size(vidFrames1_4,4);

positions1_4 = zeros(2, numFrames1_4);
for j = 1:numFrames1_4
    X = vidFrames1_4(:,:,:,j); %y,x
    X((1:480),(1:250),:) = zeros(480,250,3);
    X((1:480),(451:640),:) = zeros(480,190,3);
    X((1:200),(1:640),:) = zeros(200,640,3);
    %keep red component
    X(:,:,(2:3)) = zeros(480,640, 2);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions1_4(:, j) = [bigx; bigy];
end

load('cam2_4.mat')
numFrames2_4 = size(vidFrames2_4,4);

positions2_4 = zeros(2, numFrames2_4-13);
for j = 11:numFrames2_4-3
    [bw,X] = createMask(vidFrames2_4(:,:,:,j));
    X((1:480),(1:180),:) = zeros(480,180,3);
    X((1:480),(401:640),:) = zeros(480,240,3);
    X((401:480),(1:640),:) = zeros(80,640,3);
    X((1:130),(1:640),:) = zeros(130,640,3);
    X = rgb2gray(X);
```

```
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    X((bigy-10:bigy+9),(bigx-10:bigx+9),:) = ones(20,20)*255;
    %imshow(X)

    positions2_4(:, j-10) = [bigx; bigy];
end

load('cam3_4.mat')
numFrames3_4 = size(vidFrames3_4,4);

positions3_4 = zeros(2, numFrames3_4-2);
for j = 3:numFrames3_4
    X = vidFrames3_4(:,:,:,j); %y,x
    X((1:150),(1:640),:) = zeros(150,640,3);
    X((301:480),(1:640),:) = zeros(180,640,3);
    X((1:480),(1:220),:) = zeros(480,220,3);
    %keep red component
    X(:,:,(2:3)) = zeros(480,640, 2);
    M = max(abs(X),[],'all');
    [bigy,bigx] = ind2sub(size(X),find(abs(X) == M));
    %imshow(X)
    bigx = floor(mean(bigx));
    bigy = floor(mean(bigy));
    positions3_4(:, j-2) = [bigx; bigy];
end

position_4 = [positions1_4; positions2_4; positions3_4];
plotEverything(position_4, 392)
%%
function plotEverything(positions, data_length)
    positions = bsxfun(@minus,positions,mean(positions, 2));
    % first plot three cameras' x and y motion
    figure(1)
    tiledlayout(3,1)
    nexttile
    plot([1:data_length], positions(1,:), 'displayname', 'horizontal', 'linewidth', 3), hold on
    plot([1:data_length], positions(2,:), 'displayname', 'vertical', 'linewidth', 3)
    title('Camera_1')
    set(gca,'Fontsize',24)
    legend();
    nexttile
    plot([1:data_length], positions(3,:), 'displayname', 'horizontal', 'linewidth', 3), hold on
    plot([1:data_length], positions(4,:), 'displayname', 'vertical', 'linewidth', 3)
    title('Camera_2')
    set(gca,'Fontsize',24)
    legend();
    nexttile
    plot([1:data_length], positions(5,:), 'displayname', 'horizontal', 'linewidth', 3), hold on
    plot([1:data_length], positions(6,:), 'displayname', 'vertical', 'linewidth', 3)
    title('Camera_3')
    set(gca,'Fontsize',24)
    legend();

    % plot the singular values
    [~,n]=size(positions);
    [U,S,V] = svd(positions'/sqrt(n-1), 'econ');
```

```
    lambda =diag(S).^2;
    figure(2)
    plot(lambda / sum(lambda),'ko','Linewidth',2)
    title('Energy of diagonal variances')
    set(gca,'Fontsize',16)

    figure(3)
    Y = V' * positions;
    plot([1:data_length], Y(1, :), [1:data_length], Y(2, :), [1:data_length], Y(3, :),
        'linewidth',2)
    title('Projection of displacement data on principal components')
    set(gca,'Fontsize',16)
    legend('PC1', 'PC2','PC3', 'PC4')
end

function [BW,maskedRGBImage] = createMask(RGB)
%createMask Threshold RGB image using auto-generated code from colorThresholder app.
%  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
%  auto-generated code from the colorThresholder app. The colorspace and
%  range for each channel of the colorspace were set within the app. The
%  segmentation mask is returned in BW, and a composite of the mask and
%  original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 18-Feb-2021
%------------------------------------------------------


% Convert RGB image to chosen color space
I = rgb2hsv(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.935;
channel1Max = 0.900;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.000;
channel2Max = 1.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.000;
channel3Max = 1.000;

% Create mask based on chosen histogram thresholds
sliderBW = ( (I(:,:,1) >= channel1Min) | (I(:,:,1) <= channel1Max) ) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
BW = sliderBW;

% Invert mask
BW = ~BW;

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

end
```